Typing

with

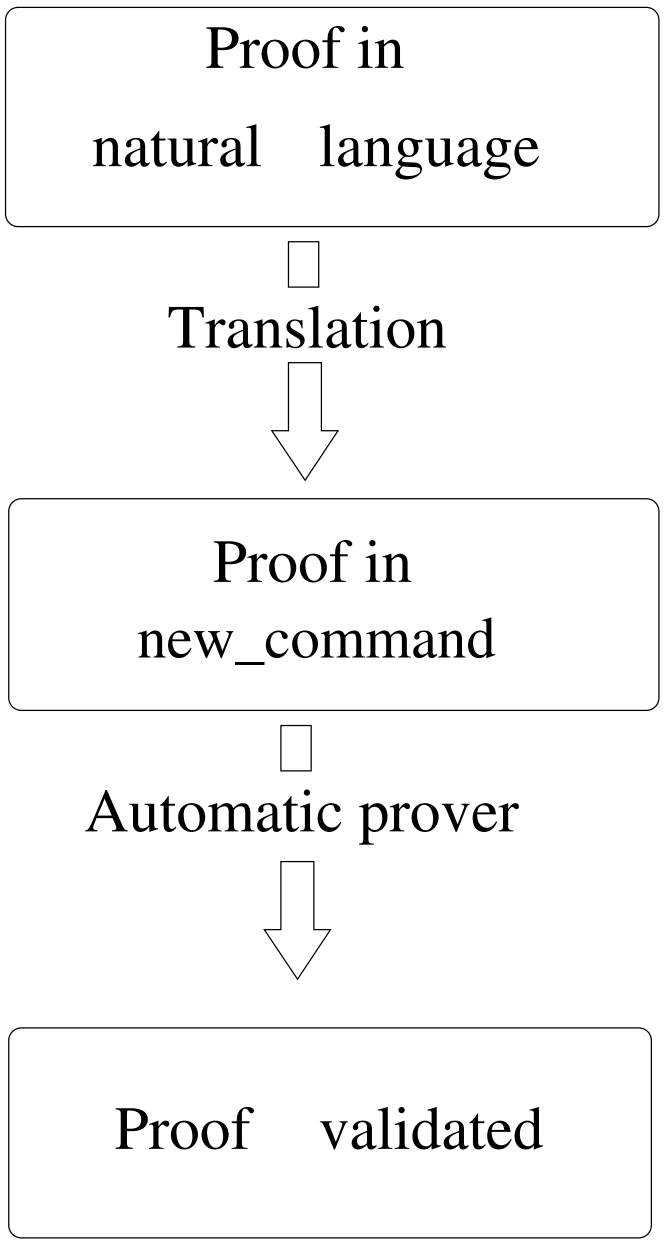
two arrows

Patrick Thévenon, PhD Student

Université de Savoie,Chambéry
Laboratoire de mathématiques,LAMA

DemoNat

- Aim of the DemoNat project :
  Analyse, validate proofs in natural language
- Teams working on this project :
  - Lattice / TaLaNa (Paris)
  - Calligramme (Nancy)
  - LaMa (Chambéry)


Proof in natural language
Translation
Proof in new_command
Automatic prover
Proof validated

Tasks

- Nancy and Paris :
  Translation from french to new_commands using the ACGs
- Chambéry :
  Definition of new_commands
  Implementation of a prover designed for this project

ACGs

Two signatures
- Σ_1 : abstract language
- Σ_2 : object language

Both are based on linear lambda calculus

$\mathcal{L}$ a lexicon from Σ_1 to Σ_2

$$\mathcal{L} : \Sigma_1 \longrightarrow \Sigma_2$$

The user gives the mapping from constants in Σ_1 to terms in Σ_2

The user does not want to give the mapping from atomic types in Σ_1 to types in Σ_2.

We would like to automaticaly have the complete lexicon thanks to the condition

$$\mathcal{L}(c) \;:\; \mathcal{L}(\tau(c))$$

In the case of simply typed linear lambda calculus it is quite easy

It could be interesting to add some more things to the calculus

- Dealing with semantics, some formulas can be of the shape $\forall x P(x) \to Q(x)$, which is no more linear. But as we want to keep linearity, we add to linear lambda-calculus intuitionistic variables, which introduces a calculus with two arrows.
- Some features could be added to the atomic types (gender with cases feminine and masculine for example)

In order to complete the lexicon we need first to be able to find the principal type of a term ($\mathcal{L}(c)$ for each constant)

In this talk we will only talk about simply typed lambda calculus with two arrows (linear and intuitionistic), which introduces some problems for the principal type

Overview

- introduction of the calculus
- the principal typing algorithm
- study of fragments
  - linear terms
  - η-long terms

The calculus

$$\frac{}{\Gamma; \vdash c : \tau(c)}$$

$$\frac{}{\Gamma; x : \gamma \vdash x : \gamma} \qquad \frac{}{\Gamma, x : \gamma; \vdash x : \gamma}$$

$$\frac{\Gamma; \Delta, x : \alpha \vdash t : \beta}{\Gamma; \Delta \vdash \mathring{\lambda} x.t : \alpha \multimap \beta} \qquad \frac{\Gamma, x : \alpha; \Delta \vdash t : \beta}{\Gamma; \Delta \vdash \lambda x.t : \alpha \to \beta}$$

$$\frac{\Gamma; \Delta_1 \vdash t : \alpha \multimap \beta \quad \Gamma; \Delta_2 \vdash u : \alpha}{\Gamma; \Delta_1, \Delta_2 \vdash (t\, u) : \beta}\ (*)$$

$$\frac{\Gamma; \Delta \vdash t : \alpha \to \beta \quad \Gamma; \vdash u : \alpha}{\Gamma; \Delta \vdash (t\, u) : \beta}$$

$(*)$ $Dom(\Delta_1) \cap Dom(\Delta_2) = \emptyset$

Typing algorithm

- typing rule scheme

$$\frac{\Gamma; \Delta \vdash t : \alpha{-?}_n\beta \quad \Gamma; \vdash u : \alpha}{\Gamma; \Delta \vdash (t\, u) : \beta}$$

- unification of types with $\multimap$, $\rightarrow$ and $-?$
  first order $\Rightarrow$ mgu

- usual typing algorithm (Damas-Milner) with constraints

- the algorithm gives a typing tree

- the principal type is the judgment at the root

Constraints

while typing application $(u\,v)$

– if v has free linear variables u must have type $\multimap$

– overwise we take a new unspecified arrow $-?$ to type u

Why do we take fragments

let

$$t = \lambda g \lambda f \overset{\circ}{\lambda} x \lambda u.(g \quad (f\ x) \quad (f\ \lambda t.(t\ u)))$$

its principal type is

$$\begin{array}{rcl} \vdash t\ :\ (b \multimap b\text{–?}_1 n) & \rightarrow & \\ (((a\text{–?}_2 e) \rightarrow e) \multimap b) & \rightarrow & \\ ((a\text{–?}_2 e) \rightarrow e) & \multimap & \\ a & \rightarrow & n \end{array}$$

t is neither linear nor η-long

Arrow property

- a typed term has the arrow property if
  - the unspecified arrows are negative
  - the intuitionistic arrows are positive

- linear terms have the arrow property
- η-long terms have the arrow property

Linear terms

- a linear term contains intuitionistic variables but they must appear once
- constants considered have the arrow property

Proposition 1 *t a typable linear term*
then
its principal type satisfies :
– *each type variable appearing appears twice*
  with a positive occurrence
  and a negative occurrence
– *the type has the arrow property*
– *the unspecified arrows are distinct*

Proof :

– prove it for normal terms
– prove that if $t \rightarrow_\beta t'$ then there exists
  S s.t. $\text{–?}_i \mapsto \multimap$ and s.t. PT(t)=S PT(t') $\square$

η-long terms

- to be η-long is a property of a typing tree
- a term is η-long if
  it has an η-long typing tree
- if a term is η-long
  Then
  it is η-long for its principal type

Typing algorithm

the typing algorithm is adapted

- types only terms of shape
  - $t = (x\, t_1 \ \ldots\ t_n)$
  - $t = (c\, t_1 \ \ldots\ t_n)$
  - $t = \lambda x.u$
  - $t = \reflectbox{$\lambda$}x.u$
  - $t = (\lambda x_1 \ldots x_n.t' \quad t_1 \ \ldots\ t_n)$
    $t' \neq \lambda x.u$ or $\reflectbox{$\lambda$}x.u$

- succeeds only if the tree is η-long

Address

- adresses c are lists of l (left) and r (right)
  - the empty address is [ ]
  - $c_1 :: c_2$ is the concatenation
  - l^k is the address with k times l

- $f(c, T)$, T a type, c an address, is the sub-type at the address c of T

Justification

- we want to justify each arrow and atom of a principal type
- we define an application φ which
  - takes
    - an address c of a type T
    - a set of terms of type T
  - gives
    a set of subterms with type $f(c, T)$
- φ is generaly defined only for η-long terms

Examples

$$t = \lambda x.(f \quad (x \quad t_1 \quad \lambda y.t_3) \quad (x \quad t_2 \quad \lambda z.t_4))$$

$$\vdash t : T = (a \to (b \to c) \to d) \to e$$

- $c_1 = [l, r] \Rightarrow f(c_1, T) = (b \to c) \to d$

  $$\varphi(c_1, \{t\}) = \{(x\ t_1), (x\ t_2)\}$$

- $c_2 = [l, r, l] \Rightarrow f(c_2, T) = (b \to c)$

  $$\varphi(c_2, \{t\}) = \{\lambda y.t_3, \lambda z.t_4\}$$

- $c_3 = [l, r, l, l] = c_2 :: l \Rightarrow f(c_3, T) = b$

  $$\begin{aligned} \varphi(c_3, \{t\}) &= \varphi([l], \{\lambda y.t_3, \lambda z.t_4\}) \\ &= \{y, z\} \end{aligned}$$

Back to typing

While typing $t = (x\, t_1\ \ldots\ t_n)$

- get the principal types of the t_i
- unify the types of the variables
  see that as applying substitutions to
  a set of judgments with same free variables
- unify the type of x obtained with the type of
  x built with the types of the t_i
  see that as applying substitutions to
  a set of two judgments

Points

in a set of judgments

- a variable point is the choice of
  - a judgment
  - a free variable of the judgment
  - an address in the type of the variable

- a term point is the choice of
  - a judgment (a term)
  - an address in the type of the term

- What is interesting in a point is the head of the sub-type designed by the address

Classes

in a set of judgments

- the class of a variable point (t, x, c)
  is $(l :: c, \{\lambda x.t_i\})$ with t_i s.t.
  x has the same type head at the address c

- the class of a term point (t, c) is $(c, \{t\})$

- a class is justified if φ is defined

Examples

$t_1 = \lambda y(x\ y)$
$t_2 = \lambda w \parr z(x \quad (w\ z))$

$x : a\text{–?}b \vdash t_1 : a \to b$
$x : a \multimap b \vdash t_2 : (c \multimap a) \to c \multimap b$

$(t_1, x, [])$, $(t_1, x, [l])$ and $(t_2, [l, r])$
are points whose classes are

$([l], \{\lambda x.t_1\})$, $([l, l], \{\lambda x.t_1, \lambda x.t_2\})$ and $([l, r], \{t_2\})$

Class property

– the terms have an η-long principal type
– the free variables of the terms are the same and are free in an η-long term t
– each class Cl is justified
– if the justifying terms are variables x of $\lambda x.u$ s.t. $x \notin u$ then the type is an atom a and a is unique
– the unspecified arrows are unique and negative
– the $\rightarrow$ are positive

Lemma 1 *The class property is stable by the unification of the types of the variables*

Proposition

Proposition 2 *t a term s.t. the algorithm gives an η-long typing tree for t then the root of the tree satisfies*

– *each point P is justified*
– *if the justifying terms are variables x of $\lambda x.u$ s.t. $x \notin u$ then the type is an atom a and a is unique*
– *unspecified arrow are unique and negative*
– *the $\rightarrow$ are positive*

Corollary 1 *let t η-long term of type T
if T has a negative $\rightarrow$
then
this arrow can be replaced by a $\multimap$*

Proof of corollary

the principal type of t is η-long

$\exists S$ s.t. $SPT{=}T$

PT has no negative arrow $\rightarrow$

S changes

– a $-$? into $\rightarrow$:
  $-$? is unique
– a type variable α into a type A containing $\rightarrow$:
  we prove α is unique because it is justified by variables x of $\lambda x.u$ with $x \notin u$.

Future work

- Proof quite complex, very technical, we would like to find something more readable. Maybe there could be a solution with sub-typing (where $\multimap \subset \rightarrow$)
- Solve the matching problem for the calculus with two arrows
- Add features to atomic types, in order to avoid the multiplication of entries into the signatures.
- In the typed calculus with features, find an algorithm to complete the lexicon